**Capstone Project – Myntra Fashion Sales Data Analysis**

**Problem Statement**

The dataset "Myntra Fashion Dataset" provides information on products listed on the e-commerce platform Myntra.com, including Product\_id, BrandName, Category, Individual\_category, category\_by\_Gender, Description, DiscountPrice (in Rs), OriginalPrice (in Rs), DiscountOffer, SizeOption, Ratings and Reviews. The objective of this analysis is to leverage the dataset to analyze trends in product popularity, pricing, and discounts, and develop predictive models for product recommendations and pricing strategies. Specifically, the goals are:

1. Conduct exploratory data analysis to understand the distribution of product prices, ratings, and discounts across different categories and brands.

2. Identify trends in product popularity based on ratings and total rating counts.

3. Analyze the impact of discounts on product sales and revenue.

4. Develop a recommendation system using machine learning to suggest products to users based on their preferences and browsing history.

5. Build predictive models to forecast product prices and discounts based on historical data and market trends.

This analysis is important for Myntra and other e-commerce platforms to optimize their product offerings, pricing strategies, and marketing efforts. By understanding trends in product popularity and pricing, businesses can make informed decisions to improve customer satisfaction and drive sales.

**Introduction**

Myntra, a leading player in Indian e-commerce, excels as a fashion hub offering diverse products and personalized experiences.

However, in a rapidly evolving market, it faces the challenge of optimizing sales and product offerings. This project aims to leverage data insights to refine Myntra's sales strategy, ensuring it remains competitive and responsive to customer needs.

Through analysis of sales data, predictive modeling, and actionable insights, we strive to enhance Myntra's performance and elevate the shopping experience for its customers.

**Business Problem**

* Myntra aims to boost sales and optimize product offerings amidst intense market competition.
* Key challenges include understanding customer preferences, optimizing product assortment, refining pricing strategies, enhancing personalization, and improving inventory management.
* By leveraging data-driven insights, Myntra seeks to drive revenue growth, improve customer satisfaction, and maintain its market leadership position.

**Approach of solving the Business problem**

To enhance sales performance and optimize product offerings for Myntra, we employ a systematic, data-driven approach

* **Understanding the Business Problem** - We align with Myntra's objectives and analyze its market positioning and customer demographics to tailor our solutions.
* **Data Collection** - We gather comprehensive sales data from Myntra's database, including product attributes, transactions, demographics, pricing, and discounts.
* **Data Preprocessing** - We clean and preprocess the data to ensure accuracy and completeness, handling missing values, duplicates, and standardizing formats.
* **Exploratory Data Analysis (EDA)** - Through descriptive statistics and visualization, we uncover trends and relationships, identifying key drivers of sales performance and customer preferences.
* **Modeling** - Leveraging insights from EDA, we develop predictive models using machine learning algorithms to forecast sales and optimize product recommendations.
* **Evaluation** - We assess model performance using metrics and validation techniques, ensuring the reliability and generalizability of recommendations.

Our emphasis on data-driven decision-making empowers Myntra to understand customer behavior, identify opportunities, and optimize strategies, ultimately maximizing sales potential and delivering value to customers.

**Steps**

* **Data Collection** - We gather comprehensive sales data from Myntra's database, including product attributes, transactions, demographics, pricing, and discounts.
* **Data Cleaning** - Processed the Myntra Products Dataset to address missing values, duplicates, and inconsistencies, ensuring data integrity and reliability.
* **Exploratory Data Analysis** (EDA) - Conducted thorough analysis and visualization of the dataset to uncover trends, patterns, and relationships, providing valuable insights into product pricing, customer preferences, and market trends.
* **Data Encoding** - Transformed categorical variables into numerical representations using techniques like one-hot encoding or label encoding, enabling machine learning algorithms to process the data effectively.
* **Data Splitting** - Partitioned the dataset into training and testing sets to facilitate model training and evaluation, ensuring unbiased performance assessment.
* **Model Selection** - Explored and evaluated different machine learning models such as linear regression, decision trees, and random forests to identify the most suitable algorithm for predicting product prices on Myntra.
* **Model Prediction** - Trained the selected model on the training data to learn the underlying patterns and relationships, enabling accurate predictions of product prices.
* **Model Evaluation** - Evaluated the trained model's performance on the testing data using metrics such as mean squared error (MSE) or R-squared, assessing its ability to generalize to unseen data.

The analysis and modeling efforts provide valuable insights and predictive capabilities for optimizing product pricing on Myntra, ultimately enhancing sales performance and customer satisfaction. Further refinements and iterations can lead to even more accurate predictions and informed decision-making processes.

**1. Data Collection**

Data Source Link - <https://www.kaggle.com/datasets/manishmathias/myntra-fashion-dataset>

**Overview of the Dataset**

* **Size** - The dataset contains 40,274 entries and 13 columns.
* **Columns** - URL, Product\_id, BrandName, Category, Individual\_category, category\_by\_Gender, Description, DiscountPrice (in Rs), OriginalPrice (in Rs), DiscountOffer, SizeOption, Ratings, Reviews.
* **Data Types** - object (string), integer, and float.
* **Missing Values** - Columns DiscountPrice, DiscountOffer, SizeOption, Ratings, and Reviews have missing values.
* **Memory Usage** - Approximately 4.0+ MB.

**Code Explanation**

data = pd.read\_csv('/content/Myntra Fasion Clothing.csv')

This line of code reads the CSV file "Myntra Fasion Clothing.csv" located at the specified path ("/content/") into a pandas DataFrame named "data".

data.info()

The `data.info()` method provides a concise summary of the DataFrame "data," including information about the index, column data types, and non-null values present in each column. This summary helps in understanding the structure and completeness of the dataset.

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 526564 entries, 0 to 526563

Data columns (total 13 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 URL 526564 non-null object

1 Product\_id 526564 non-null int64

2 BrandName 526564 non-null object

3 Category 526564 non-null object

4 Individual\_category 526564 non-null object

5 category\_by\_Gender 526564 non-null object

6 Description 526564 non-null object

7 DiscountPrice (in Rs) 333406 non-null float64

8 OriginalPrice (in Rs) 526564 non-null float64

9 DiscountOffer 452258 non-null object

10 SizeOption 526564 non-null object

11 Ratings 190412 non-null float64

12 Reviews 190412 non-null float64

dtypes: float64(4), int64(1), object(8)

memory usage: 52.2+ MB

data.head()

The `data.head()` method displays the first few rows of the DataFrame "data," providing a quick glimpse of the dataset's contents. This allows for a preliminary examination of the data and helps in understanding its structure and format.

data.sample(5)

The `data.sample(5)` method randomly selects and displays 5 rows from the DataFrame "data". This provides a random sample of the dataset, allowing for a diverse view of the data and aiding in exploratory analysis.

data.shape

(526564, 13)

The `data.shape` attribute returns a tuple representing the dimensions of the DataFrame "data", where the first element corresponds to the number of rows and the second element corresponds to the number of columns. This provides information about the size or extent of the dataset.

data.columns

The `data.columns` attribute returns a list of column names present in the DataFrame "data". This provides insight into the variables or features included in the dataset, facilitating data manipulation and analysis.

Index(['URL', 'Product\_id', 'BrandName', 'Category', 'Individual\_category',

'category\_by\_Gender', 'Description', 'DiscountPrice (in Rs)',

'OriginalPrice (in Rs)', 'DiscountOffer', 'SizeOption', 'Ratings',

'Reviews'],

dtype='object')

data.describe()

The `data.describe()` method generates descriptive statistics summarizing the central tendency, dispersion, and shape of the numerical variables in the DataFrame "data". This includes metrics such as count, mean, standard deviation, minimum, 25th percentile (Q1), median (50th percentile or Q2), 75th percentile (Q3), and maximum values. It provides a quick overview of the distribution of numerical data, aiding in data exploration and understanding.

**2. Data Cleaning**

* **Identifying Missing Values** - Utilize pandas' .isnull() or .info() methods to identify columns with missing values.
* **Numeric Variables** - Fill missing values with appropriate measures of central tendency such as mean or median using pandas' .fillna() method.
* **Categorical Variables** - Fill missing values with the mode (most frequent value) or a placeholder value (e.g., "Unknown") using .fillna() method.

**Code Explanation**

data.isnull().sum()

The `data.isnull().sum()` expression computes the number of missing values (null values) for each column in the DataFrame "data". It returns a Series containing the count of null values for each column, allowing for quick identification of columns with missing data. This information is essential for data cleaning and preprocessing tasks.

URL 0

Product\_id 0

BrandName 0

Category 0

Individual\_category 0

category\_by\_Gender 0

Description 0

DiscountPrice (in Rs) 193158

OriginalPrice (in Rs) 0

DiscountOffer 74306

SizeOption 0

Ratings 336152

Reviews 336152

dtype: int64

* DiscountPrice (in Rs): 193,158 missing values.
* DiscountOffer: 74,306 missing values.
* Ratings: 336,152 missing values.
* Reviews: 336,152 missing values.

# Fill missing values in Product\_id with a placeholder value (e.g., -1)

data['Product\_id'].fillna(-1, inplace=True)

# Fill missing values in BrandName, Category, Individual\_category, category\_by\_Gender, and Description with 'Unknown'

data['BrandName'].fillna('Unknown', inplace=True)

data['Category'].fillna('Unknown', inplace=True)

data['Individual\_category'].fillna('Unknown', inplace=True)

data['category\_by\_Gender'].fillna('Unknown', inplace=True)

data['Description'].fillna('Unknown', inplace=True)

# Fill missing values in DiscountPrice with 0

data['DiscountPrice (in Rs)'].fillna(0, inplace=True)

# Fill missing values in DiscountOffer with 'No Discount'

data['DiscountOffer'].fillna('No Discount', inplace=True)

# Fill missing values in SizeOption with 'Not Specified'

data['SizeOption'].fillna('Not Specified', inplace=True)

# Fill missing values in Ratings and Reviews with the mean value

mean\_ratings = data['Ratings'].mean()

data['Ratings'].fillna(mean\_ratings, inplace=True)

mean\_reviews = data['Reviews'].mean()

data['Reviews'].fillna(mean\_reviews, inplace=True)

# Check for missing values after handling

missing\_values\_count = data.isnull().sum()

print("Remaining missing values count:\n", missing\_values\_count)

* Product\_id: Missing values are filled with a placeholder value (-1).
* BrandName, Category, Individual\_category, category\_by\_Gender, and Description: Missing values in these categorical columns are filled with the string 'Unknown'.
* DiscountPrice (in Rs): Missing values are filled with 0.
* DiscountOffer: Missing values are filled with 'No Discount'.
* SizeOption: Missing values are filled with 'Not Specified'.
* Ratings and Reviews: Missing values in these numerical columns are filled with the mean value of their respective columns.

**3. EDA**

**Top 10 Brands based on Average Price of Products**

# Assuming you have a DataFrame named 'df' containing your data

top\_10\_brands\_price\_based = data.groupby('BrandName')['OriginalPrice (in Rs)'].mean().nlargest(10)

# Plotting the top 10 brands based on average price

plt.figure(figsize=(8, 6))

top\_10\_brands\_price\_based.plot(kind='bar', color='green')

plt.title('Top 10 Brands based on Average Price of Products')

plt.xlabel('Brand')

plt.ylabel('Average Price (in Rs)')

plt.xticks(rotation=90, ha='right')

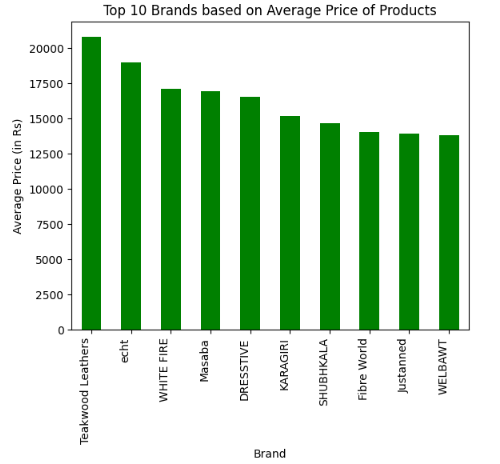
plt.tight\_layout()

plt.show()

**Code Explanation**

* It groups the data in the DataFrame 'data' by the 'BrandName' column and calculates the mean of the 'OriginalPrice (in Rs)' column for each brand.
* It selects the top 10 brands with the highest average prices using the nlargest() method.
* It plots a bar chart depicting the top 10 brands based on their average product prices.
* The plot is customized with appropriate titles, labels, and formatting to enhance readability.
* Finally, it displays the plot using Matplotlib.

**Output**

****

**Relationship between Price and Ratings**

# Relationship between price and ratings

plt.figure(figsize=(10, 6))

sns.scatterplot(x='OriginalPrice (in Rs)', y='Ratings', data=data, color='green')

plt.title('Relationship between Price and Ratings')

plt.xlabel('Price (in Rs)')

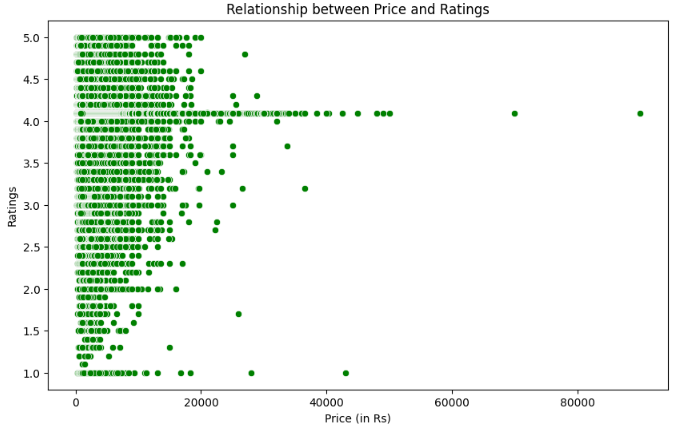
plt.ylabel('Ratings')

plt.show()

**Code Explanation**

* It utilizes Seaborn's scatterplot function to create a scatter plot.
* The x-axis represents the product's original price (in Rs), and the y-axis represents the product's ratings.
* Each data point in the scatter plot represents a product, with its position determined by its price and rating.
* The plot is customized with appropriate titles, labels, and colors for better visualization.
* Finally, the plot is displayed using Matplotlib.

**Output**

****

**Top Ten Brands based on Average Ratings**

# Calculate the average ratings for each brand

average\_ratings\_by\_brand = data.groupby('BrandName')['Ratings'].mean()

# Sort brands based on average ratings and select the top ten

top\_ten\_brands\_ratings\_based = average\_ratings\_by\_brand.nlargest(10)

# Plot the top ten brands based on average ratings

plt.figure(figsize=(12, 6))

sns.barplot(x=top\_ten\_brands\_ratings\_based.values, y=top\_ten\_brands\_ratings\_based.index, palette='viridis')

plt.xlabel('Average Ratings')

plt.ylabel('Brand')

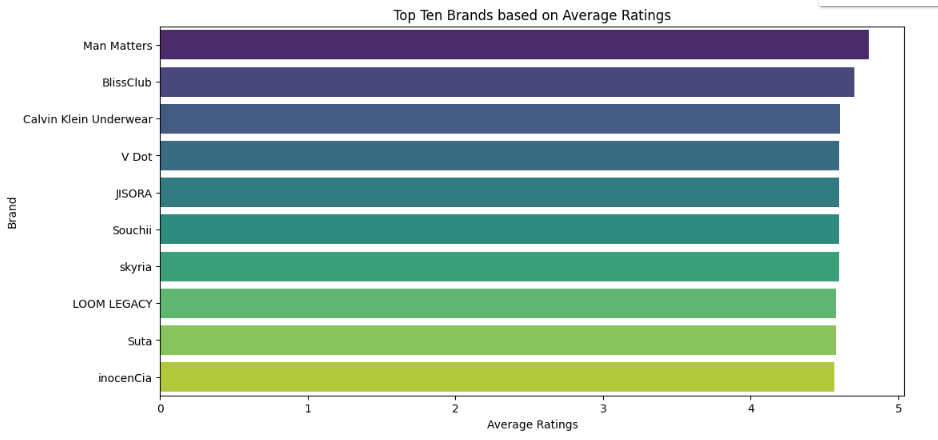
plt.title('Top Ten Brands based on Average Ratings')

plt.show()

**Code Explanation**

* It computes the average ratings for each brand by grouping the data by the 'BrandName' column and calculating the mean of the 'Ratings' column.
* It sorts the brands based on their average ratings using the nlargest() method.
* It creates a horizontal bar plot using Seaborn's barplot function, with the x-axis representing the average ratings and the y-axis representing the brand names.
* The plot is customized with appropriate labels and titles for better interpretation.
* Finally, the plot is displayed using Matplotlib.

**Output**

****

**Top Ten Brands based on Total Reviews**

# Calculate the total reviews for each brand

total\_reviews\_by\_brand = data.groupby('BrandName')['Reviews'].sum()

# Sort brands based on total reviews and select the top ten

top\_ten\_brands\_reviews\_based = total\_reviews\_by\_brand.nlargest(10)

# Plot the top ten brands based on total reviews

plt.figure(figsize=(12, 6))

sns.barplot(x=top\_ten\_brands\_reviews\_based.values, y=top\_ten\_brands\_reviews\_based.index, palette='Oranges\_r')

plt.xlabel('Total Reviews')

plt.ylabel('Brand')

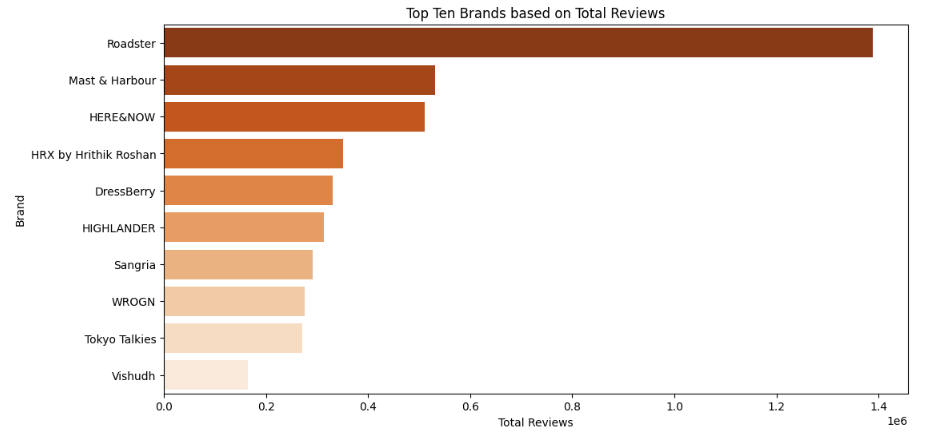
plt.title('Top Ten Brands based on Total Reviews')

plt.show()

**Code Explanation**

* It computes the total reviews for each brand by grouping the data by the 'BrandName' column and summing the values of the 'Reviews' column.
* It sorts the brands based on their total reviews using the nlargest() method.
* It creates a horizontal bar plot using Seaborn's barplot function, with the x-axis representing the total reviews and the y-axis representing the brand names.
* The plot is customized with appropriate labels and titles for better interpretation.
* Finally, the plot is displayed using Matplotlib.

**Output**

****

**Distribution of Categories**

# Count the occurrences of each category

category\_counts = data['Category'].value\_counts()

# Plot a pie chart for the distribution of categories

plt.figure(figsize=(8, 6))

plt.pie(category\_counts, labels=category\_counts.index, autopct='%1.1f%%', startangle=140)

plt.title('Distribution of Categories\n')

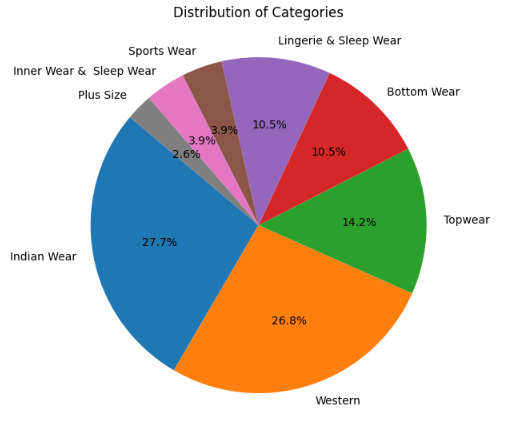
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()

**Code Explanation**

* It uses the value\_counts() method to count the occurrences of each category in the 'Category' column.
* It creates a pie chart using Matplotlib's pie() function, with the category names as labels and the corresponding counts as values.
* The autopct='%1.1f%%' parameter formats the percentage labels on the pie chart to one decimal place.
* The startangle=140 parameter specifies the starting angle for the first wedge of the pie chart.
* The plt.title() function sets the title of the plot.
* The plt.axis('equal') function ensures that the pie chart is drawn as a circle.
* Finally, the plot is displayed using Matplotlib.

**Output**

****

import seaborn as sns

# Plot a count plot for the distribution of categories by gender

plt.figure(figsize=(8, 6))

sns.countplot(x='category\_by\_Gender', data=data, palette='pastel')

plt.title('Distribution of Categories by Gender')

plt.xlabel('Category by Gender')

plt.ylabel('Count')

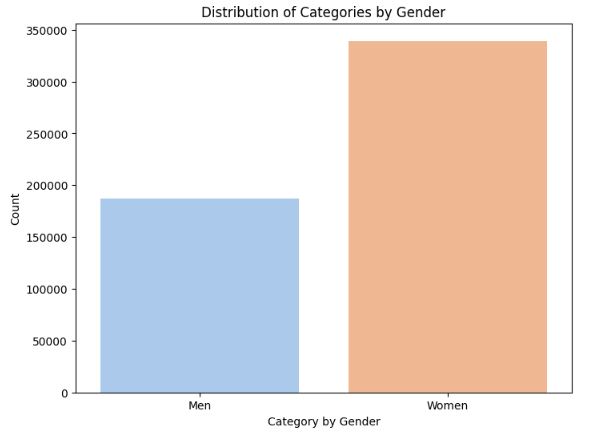
plt.xticks(rotation=360)

plt.show()

**Code Explanation**

* It specifies the x-axis variable as 'category\_by\_Gender' and the DataFrame as 'data'.
* Seaborn's countplot function automatically counts the occurrences of each category by gender and visualizes them as bars.
* The palette='pastel' parameter sets the color palette for the plot, giving it a soft and visually appealing appearance.
* The plot is customized with appropriate titles, labels, and rotation of x-axis tick labels for better readability.
* Finally, the plot is displayed using Matplotlib.

**Output**

****

**Distribution of Product Prices by Category**

# Distribution of product prices across different categories and brands

plt.figure(figsize=(8, 5))

sns.boxplot(x='Category', y='OriginalPrice (in Rs)', data=data)

plt.title('Distribution of Product Prices by Category')

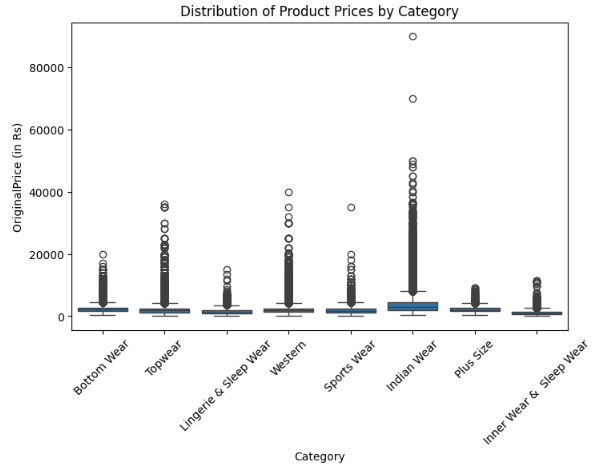
plt.xticks(rotation=45)

plt.show()

**Code Explanation**

* It specifies the x-axis variable as 'Category' and the y-axis variable as 'OriginalPrice (in Rs)'.
* Seaborn's boxplot function generates a box plot for each category, showing the distribution of product prices within each category.
* The box plot visualizes the median, quartiles, and potential outliers in the distribution of product prices for each category.
* The plt.title() function sets the title of the plot, and plt.xticks(rotation=45) rotates the x-axis tick labels for better readability.
* Finally, the plot is displayed using Matplotlib.

**Output**

****

**Correlation Heatmap**

# Encode categorical variables if needed

encoded\_data = data.copy()

encoded\_data['Category'] = encoded\_data['Category'].astype('category').cat.codes

encoded\_data['Individual\_category'] = encoded\_data['Individual\_category'].astype('category').cat.codes

encoded\_data['category\_by\_Gender'] = encoded\_data['category\_by\_Gender'].astype('category').cat.codes

encoded\_data['SizeOption'] = encoded\_data['SizeOption'].astype('category').cat.codes

encoded\_data['DiscountOffer'] = encoded\_data['DiscountOffer'].astype('category').cat.codes

# Calculate the correlation matrix

correlation\_matrix = encoded\_data.corr()

# Plot the heatmap

plt.figure(figsize=(8,6))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

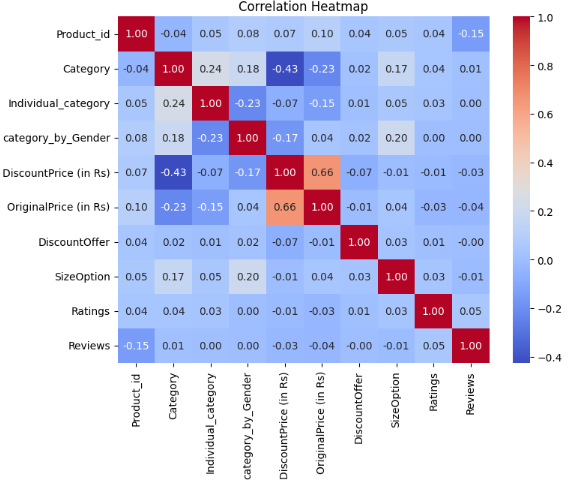
plt.title('Correlation Heatmap')

plt.show()

**Code Explanation**

* It creates a copy of the original DataFrame 'data' named 'encoded\_data'.
* Categorical variables like 'Category', 'Individual\_category', 'category\_by\_Gender', 'SizeOption', and 'DiscountOffer' are encoded using the astype('category').cat.codes method, which assigns numerical codes to each category.
* The correlation matrix is computed for the encoded data using the corr() method, which calculates the pairwise correlations between numerical variables.
* Seaborn's heatmap function is used to plot the correlation matrix as a heatmap, with annotations displaying the correlation coefficients.
* The heatmap is customized with a title, colormap ('coolwarm'), and formatting of the correlation coefficients to two decimal places.

**Output**

****

**Goal 3: Analyze the impact of discounts on product sales and revenue**

# Calculate the total revenue for each product (considering discounted prices)

data['TotalRevenue'] = data['DiscountPrice (in Rs)'] \* data['Reviews']

# Plot the relationship between discounts and total revenue

plt.figure(figsize=(10, 6))

sns.scatterplot(x='DiscountPrice (in Rs)', y='TotalRevenue', data=data, color='green', alpha=0.5)

plt.title('Impact of Discounts on Product Revenue')

plt.xlabel('Discounted Price (in Rs)')

plt.ylabel('Total Revenue')

plt.show()

# Plot the relationship between discounts and total sales (total rating counts)

plt.figure(figsize=(10, 6))

sns.scatterplot(x='DiscountPrice (in Rs)', y='Reviews', data=data, color='purple', alpha=0.5)

plt.title('Impact of Discounts on Product Sales')

plt.xlabel('Discounted Price (in Rs)')

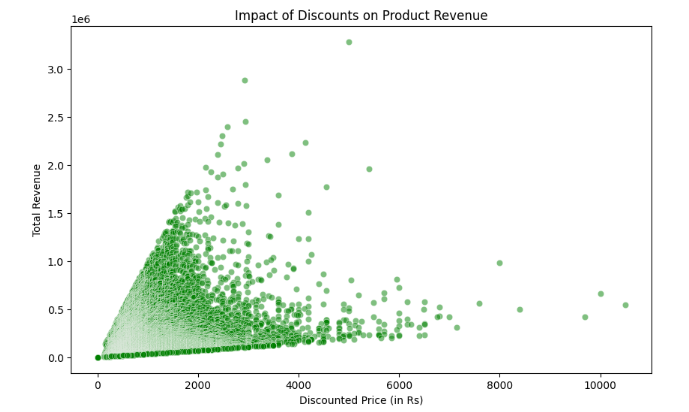
plt.ylabel('Total Sales (Total Rating Counts)')

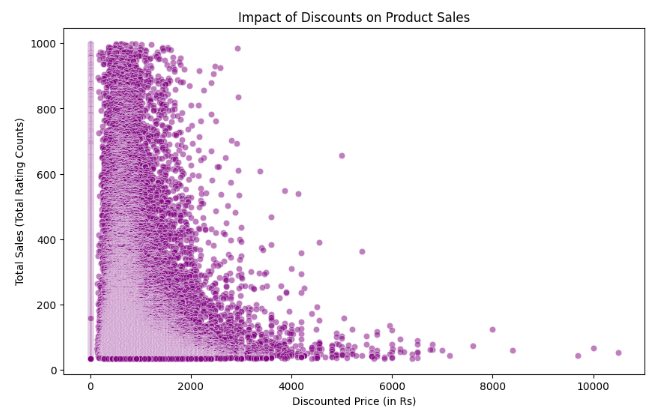
plt.show()

**Code Explanation**

* It calculates the total revenue for each product by multiplying the discounted price ('DiscountPrice (in Rs)') by the total number of reviews ('Reviews'), assuming that each review corresponds to a sale.
* It plots a scatter plot to visualize the relationship between discounted prices and total revenue. The x-axis represents the discounted price, while the y-axis represents the total revenue. Each data point in the scatter plot represents a product, with its position determined by its discounted price and total revenue.
* It plots another scatter plot to visualize the relationship between discounted prices and total sales (total rating counts). The x-axis represents the discounted price, while the y-axis represents the total sales, measured by the total rating counts.
* Both scatter plots are customized with appropriate titles, labels, and colors for better visualization.

**Output**

****

****

**Goal 4 - Develop a recommendation system using machine learning to suggest products to users based on their preferences and browsing history.**

# Install the Surprise library

# !pip install scikit-surprise

from surprise import Reader, Dataset, SVD

from surprise.model\_selection import cross\_validate, train\_test\_split

# Load the dataset

reader = Reader(rating\_scale=(1, 5))

data\_surprise = Dataset.load\_from\_df(data[['Product\_id', 'Reviews', 'Ratings']], reader)

# Split the data into train and test sets

trainset, testset = train\_test\_split(data\_surprise, test\_size=0.2, random\_state=42)

# Use Singular Value Decomposition (SVD) algorithm for recommendation

model = SVD()

model.fit(trainset)

# Generate predictions for the test set

predictions = model.test(testset)

# Evaluate the model

from surprise import accuracy

accuracy.rmse(predictions)

# Make recommendations for a specific user

user\_id = 123  # Example user ID

user\_items = data[data['Product\_id'] == user\_id]['Product\_id'].values.tolist()

user\_unseen\_items = [item for item in data['Product\_id'].unique() if item not in user\_items]

# Predict ratings for unseen items

user\_ratings = [(user\_id, item\_id, model.predict(user\_id, item\_id).est) for item\_id in user\_unseen\_items]

# Sort the recommendations by predicted ratings

sorted\_recommendations = sorted(user\_ratings, key=lambda x: x[2], reverse=True)

# Get top N recommendations

top\_n = 10

top\_recommendations = sorted\_recommendations[:top\_n]

# Print the top N recommendations

print(f"Top {top\_n} Recommendations for User {user\_id}:")

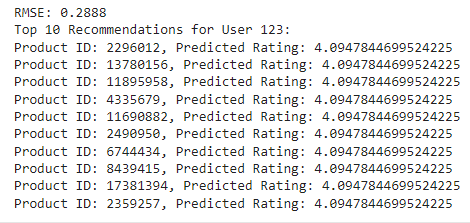
for item in top\_recommendations:

    print(f"Product ID: {item[1]}, Predicted Rating: {item[2]}")

**Code Explanation**

* It loads the Surprise library and necessary modules.
* The dataset is loaded using the Surprise Reader and Dataset classes, specifying the rating scale.
* The data is split into train and test sets using the train\_test\_split function.
* The Singular Value Decomposition (SVD) algorithm is used for recommendation by initializing the SVD model and fitting it to the training data.
* Predictions are generated for the test set using the trained model.
* The root mean squared error (RMSE) is calculated to evaluate the performance of the model.
* Recommendations are generated for a specific user by predicting ratings for unseen items and sorting them based on predicted ratings.
* The top N recommendations are printed for the user.

**Output**

****

**Model Prediction and Evaluation**

**Linear Regression**

model = LinearRegression(n\_jobs=-1)

model.fit(X\_train,y\_train)

**Code Explanation**

* The Linear Regression model is initialized with the LinearRegression class from scikit-learn.
* The n\_jobs parameter is set to -1, which indicates that the model should use all available CPU cores for computation.
* The fit method is called on the model object, passing the training data X\_train and corresponding target values y\_train as arguments.

y\_test\_predict = model.predict(X\_test)

print('Model\_Test\_Accuracy:',r2\_score(y\_test,y\_test\_predict))

y\_val\_predict = model.predict(X\_val)

print('Model\_Validation\_Accuracy:',r2\_score(y\_val,y\_val\_predict))

**Result**



**KNN Model**

from sklearn.neighbors import KNeighborsRegressor

model = KNeighborsRegressor(n\_jobs=-1)

model.fit(X\_train,y\_train)

**Code Explanation**

* The KNeighborsRegressor model is initialized with the KNeighborsRegressor class from scikit-learn.
* The n\_jobs parameter is set to -1, indicating that the model should utilize all available CPU cores for computation.
* The fit method is called on the model object, passing the training data X\_train and corresponding target values y\_train as arguments.

y\_test\_predict = model.predict(X\_test)

print('Model\_Test\_Accuracy:',r2\_score(y\_test,y\_test\_predict))

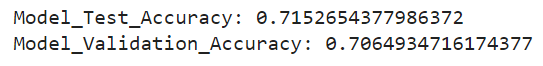
y\_val\_predict = model.predict(X\_val)

print('Model\_Validation\_Accuracy:',r2\_score(y\_val,y\_val\_predict))

**Code Explanation**

* The predict method is used to generate predictions (y\_test\_predict and y\_val\_predict) for the test (X\_test) and validation (X\_val) datasets, respectively.
* The r2\_score function from scikit-learn is applied to calculate the R-squared score, which measures the proportion of the variance in the dependent variable that is predictable from the independent variables.
* The R-squared score for the test dataset is printed as "Model\_Test\_Accuracy:", and the R-squared score for the validation dataset is printed as "Model\_Validation\_Accuracy:".

**Result**



**Randam Forest**

model = RandomForestRegressor(n\_jobs=-1)

model.fit(X\_train,y\_train)

**Code Explanation**

* The RandomForestRegressor model is initialized with the RandomForestRegressor class from scikit-learn.
* The n\_jobs parameter is set to -1, indicating that the model should utilize all available CPU cores for computation.
* The fit method is called on the model object, passing the training data X\_train and corresponding target values y\_train as arguments.

y\_test\_predict = model.predict(X\_test)

print('Model\_Test\_Accuracy:',r2\_score(y\_test,y\_test\_predict))

y\_val\_predict = model.predict(X\_val)

print('Model\_Validation\_Accuracy:',r2\_score(y\_val,y\_val\_predict))

**Code Explanation**

* The predict method is used to generate predictions (y\_test\_predict and y\_val\_predict) for the test (X\_test) and validation (X\_val) datasets, respectively.
* The r2\_score function from scikit-learn is applied to calculate the R-squared score, which measures the proportion of the variance in the dependent variable that is predictable from the independent variables.
* The R-squared score for the test dataset is printed as "Model\_Test\_Accuracy:", and the R-squared score for the validation dataset is printed as "Model\_Validation\_Accuracy:"

**Result**



**Reference**

**Github -** [**https**](https)[**://github.com/jananishreeSD/Myntra-Fashion/**](https://github.com/jananishreeSD/Myntra-Fashion/)

**Dataset -** [**https**](https)[**://www.kaggle.com/datasets/manishmathias/myntra-fashion-dataset**](https://www.kaggle.com/datasets/manishmathias/myntra-fashion-dataset)

**Thank You!**

**Janani Shree S D**